Appendix

```
#
# ---------------------------------------------------------------------------
#Python 2.7.2
# GeoMaskerV3.0 /2017-07-18/
# http://idv.sinica.edu.tw/tachien/geomasker
# remove folder "C:/geomasker2017/output/" before executing the script
# ---------------------------------------------------------------------------
# Import arcpy module

import timeit
import arcpy
import os

arcpy.env.overwriteOutput = True
newpath = "C:/geomasker2017/output/"
os.makedirs(newpath)
arcpy.env.workspace = "C:/geomasker2017/output"
start_time = timeit.default_timer()

# Specify input datasets: points, polygons(both are projected)
# GS=grid size
# kano=pre-specified minimum K-anonymity (>=2)

def setupMask():
    OriginalPoint = arcpy.GetParameterAsText(0)
    BasePolygon = arcpy.GetParameterAsText(1)
    fieldName=arcpy.GetParameterAsText(2)
    GS=arcpy.GetParameter(3)
    kano=arcpy.GetParameter(4)
    gmp = arcpy.GetParameterAsText(5)
    Mask(OriginalPoint, BasePolygon, fieldName, GS, kano, gmp)

def Mask(OriginalPoint, BasePolygon, fieldName, GS, kano, gmp):

 size = GS
 d_shp = OriginalPoint
 k_shp = BasePolygon
 output = "C:/geomasker2017/output"
```

```
merge = "merge.shp"
mp = gmp

# Create feature class "merge" to collect masking points
arcpy.CreateFeatureclass_management(output, merge, "POINT", "", "DISABLED", "DISABLED",
"", "", "0", "0", "0")

count = 1
loop = 0
while (count > 0):

 f_shp = newpath + "f_%s.shp" % GS
 kcopy_shp = "kcopy_%s.shp" % GS
 fc_shp = "fc_%s.shp" % GS
 f_p_shp = "f_p_%s.shp" % GS
 f_p_c_s_shp = "f_p_c_s_%s.shp" % GS
 f_p_c_shp = "f_p_c_%s.shp" % GS
 p_f_shp = "p_f_%s.shp" % GS
 p_f_s_shp = "p_f_s_%s.shp" % GS
 p_f_s_n_shp = "p_f_s_n_%s.shp" % GS
 p_f_s_s_shp = "p_f_s_s_%s.shp" % GS
 rp_shp = "rp_%s.shp" % GS
 m_shp = "m_%s.shp" % GS
 mynext = "p_f_s_n_Layer"

# Create feature class "m_shp" as final output
 arcpy.CreateFeatureclass_management(output, m_shp, "POINT")

# Copy polygon features
 arcpy.CopyFeatures_management(k_shp, kcopy_shp, "", "0", "0", "0")

# Create Fishnet "f_shp"
 ext = arcpy.Describe(BasePolygon)
 xmin = ext.extent.XMin
 xmax = ext.extent.XMax
 ymin = ext.extent.YMin
 ymax = ext.extent.YMax
 originCoordinate = str(xmin) + " " + str(ymin)
 yAxisCoordinate = str(xmin) + " " + str(ymin + 10)
```

```
 oppositeCoorner = str(xmax) + " " + str(ymax)
 temp = newpath + kcopy_shp
 arcpy.CreateFishnet_management(f_shp, originCoordinate, yAxisCoordinate, "%s" % GS, "%s" %
GS, "0", "0", oppositeCoorner, "LABELS", "", "POLYGON")

# Add field
 arcpy.AddField_management(f_shp, "fishid", "DOUBLE", "", "", "", "", "NULLABLE",
"NON_REQUIRED", "")

# Calculate field
 arcpy.CalculateField_management(f_shp, "fishid", "[FID]", "VB", "")

# Add field
 arcpy.AddField_management(f_shp, "p", "FLOAT", "", "", "", "", "NULLABLE",
"NON_REQUIRED", "")

# Clip f_shp
 arcpy.Clip_analysis(f_shp, kcopy_shp, fc_shp, "")

# Spatial join
 arcpy.SpatialJoin_analysis(fc_shp, kcopy_shp, f_p_shp, "JOIN_ONE_TO_ONE", "KEEP_ALL",
"", "INTERSECT", "", "")

# Delete field
 arcpy.DeleteField_management(f_p_shp, "Join_Count;TARGET_FID")

# Calculate field "p"="[Density]*(%GS%^2)" (total population counts in a fishnet grid)
 arcpy.CalculateField_management(f_p_shp, "p", "[%s]*%s^2" % (fieldName, GS), "", "")

# Spatial join
 fieldmappings = arcpy.FieldMappings()
 fieldmappings.addTable(f_p_shp)
 fieldmappings.addTable(d_shp)
 keepers = ["p", "Join_Count","fishid"]
 for field in fieldmappings.fields:
   if field.name not in keepers:
       fieldmappings.removeFieldMap(fieldmappings.findFieldMapIndex(field.name))

 arcpy.SpatialJoin_analysis(f_p_shp, d_shp, f_p_c_shp, "JOIN_ONE_TO_ONE",
```

```
"KEEP_COMMON", fieldmappings, "INTERSECT", "0.0001 Meters", "")

 # Add field
  arcpy.AddField_management(f_p_c_shp, "p_count", "DOUBLE", "", "", "", "", "NULLABLE",
"NON_REQUIRED", "")

 # Calculate field:"p_count"="[Join_Count]"(total case number)
  arcpy.CalculateField_management(f_p_c_shp, "p_count", "[Join_Count]", "VB", "")

 # Delete field
  arcpy.DeleteField_management(f_p_c_shp, "Join_Count;TARGET_FID")

 # Spatial join
  arcpy.SpatialJoin_analysis(d_shp, f_p_c_shp, p_f_shp, "JOIN_ONE_TO_ONE",
"KEEP_COMMON", "", "INTERSECT", "0.0001 Meters", "")

 # Process: Delete field
  arcpy.DeleteField_management(p_f_shp, "Join_Count;TARGET_FID")

 # Select p_f_shp: case not meets K-anonymity
  arcpy.Select_analysis(p_f_shp, p_f_s_n_shp, "NOT ((\"p\" >= 2) AND (\"p\" >=
(\"p_count\"*%s)))" % kano)

 # Delete field
  arcpy.DeleteField_management(p_f_s_n_shp, "p;fishid;p_count")

 # Make feature layer
  arcpy.MakeFeatureLayer_management(p_f_s_n_shp, mynext, "", "", "")

 # Count unprocessing points
  result = arcpy.GetCount_management(mynext)
  countmynext = int(result.getOutput(0))

 # Select f_p_c_shp: grid meets K-anonymity
  arcpy.Select_analysis(f_p_c_shp, f_p_c_s_shp, "(\"p\" >=( \"p_count\" *%s)) AND (\"p_count\"
>= 1)" % kano)

 # Create random points
  arcpy.CreateRandomPoints_management(output, rp_shp, f_p_c_s_shp, f_p_c_s_shp, "p_count",
```

```
"0 Meters", "POINT", "")

 # Select p_f_shp: case meets K-anonymity
  arcpy.Select_analysis(p_f_shp, p_f_s_shp, "(\"p\" >= 2) AND (\"p\" >= (\"p_count\"*%s))" %
kano)

 # Sort
  arcpy.Sort_management(p_f_s_shp, p_f_s_s_shp, "fishid ASCENDING", "UR")

 # Join field
  arcpy.JoinField_management(rp_shp, "FID", p_f_s_s_shp, "FID", "")

 # Delete field
  arcpy.DeleteField_management(rp_shp, "CID")

  if loop == 0:

    # Process: Merge
    arcpy.CopyFeatures_management(rp_shp, merge, "", "0", "0", "0")

  if loop > 0:

    arcpy.Merge_management([rp_shp,merge], m_shp, "")
    merge=m_shp

  loop = loop + 1

  # GS increasing rate
  arcpy.AddMessage("grid size: " + str(GS))
  GS = int(size*(1+0.5*loop))
  count = countmynext
  d_shp = p_f_s_n_shp

  if count == 0:
    arcpy.CopyFeatures_management(merge, mp, "", "0", "0", "0")

  arcpy.AddMessage("loop = " + str(loop))
  arcpy.AddMessage("unprocessed points = " + str(count))
  arcpy.Delete_management("in_memory")
```

```
 elapsed = timeit.default_timer() - start_time
 arcpy.AddMessage("Elapsed time: " + str(elapsed))
 print "Elapsed time: ", elapsed

if __name__ == "__main__":
    setupMask()
```